WHUCTF2020 Crypto WriteUp

- By ii

目录

这是一份来自（并不很强的）出题人的，（自认为）对新手友好的 WriteUp，包括了对 Proof of Work 和 WHUCTF2020 Crypto 中的七道题目的讨论。

祝大家变得更强。

My Best Friend BingGe

出题背景

本题主要考察替换密码（Substitution Cipher）。没想到，一个本想用作 Crypto 签到的古典密码题，却放了 4 个 Hint 才被做出来。

读题

题目提供了密文：

```
SFKTCJ{SVSDTUPMC2YUUUUGMFPDQED4V2DDQL9BUI4CNLY!FOFOFO}
```

和密钥：

```
Let_5 H4V3 Fun p1ayin9 WHUCTF Crypt0
```

密文中包含大写字母、数字、一对大括号、一个感叹号；密钥中包括大小写字母、数字、下划线、空格。

连猜带蒙

发现密文中有一对大括号，而且最前面有六个字母，根据 flag 的格式，合理猜测 S、F、K、T、C、J 这六个字母分别对应 W、H、U、C、T、F。

注意到 T 被替换为了 C（字母表第 3 个字母），H 被替换为了 F（字母表第 6 个字母），F 被替换为了 J（字母表第 6 个字母）；而密钥中第 3 个位置是字母 t，第 6 个位置是字母 H，第 10 个位置是字母 F（不计空格）。类推可得：L 应被替换为 A，E 应被替换为 B，下划线“_”应被替换为 D，数字 5 应被替换为 E，等等。对于在密钥中出现多次的字母，只考虑第一次出现的位置。用这样的方法遍历整个密钥，可以得到：

```
1. L -> A
2. E -> B
3. T -> C
4. _ -> D
5. 5 -> E
6. H -> F
7. 4 -> G
8. V -> H
9. 3 -> I
10.F -> J
11.U -> K
12.N -> L
13.P -> M
14.1 -> N
15.A -> O
16.Y -> P
17.I -> Q
18.9 -> R
19.W -> S
20.C -> T
21.R -> U
22.0 -> V
```

上面的两列，左列是明文字符，右列是对应的密文字符。显然，密文中出现了上面右列中不包含的字符，比如字母 Y、数字；密钥中不包含的明文字符也没有得到处理。很多人做到这里就停下了，对于密文中未知对应明文的字符，选择了根据语义猜测的方法，这样做的结果是失败的。

补全密码本

可以先把密钥中不包含的字母和数字列一下：B、D、G、J、K、M、O、Q、S、X、Z、2、6、7、8。右列继续写下去的话是 W、X、Y、Z，考虑把数字 0 到 9 和下划线也放进去，可以得到：

```
B -> W
D -> X
G -> Y
J -> Z
K -> 0
M -> 1
O -> 2
Q -> 3
S -> 4
X -> 5
Z -> 6
2 -> 7
6 -> 8
7 -> 9
8 -> _
```

两部分和起来就是加密的密码表。把密文中的每个字符反向对应回去，即可得到明文。

这是一种典型的替换密码，密钥可以是一个字符串、一句话，利用密钥可以构建出密码本。

解题

一些已有的工具和网站可以实现替换密码等常见密码、编码功能，如：

```
https://github.com/gchq/CyberChef
https://cryptii.com/
```

这里也提供一个自己写的 Python 脚本：

```python
key = "Let_5 H4V3 Fun p1ayin9 WHUCTF Crypt0"
all = list(range(ord('A'), ord('Z') + 1)) + list(range(ord('0'), ord('9') + 1)) + [ord('_')]

def get_book(key):
    now = 0
    enc_book = dict()
    dec_book = dict()
    for c in key.upper():
        if c not in enc_book and ord(c) in all:
            enc_book[c] = chr(all[now])
            now += 1
    for i in all:
        c = chr(i)
        if c in enc_book:
            continue
        enc_book[c] = chr(all[now])
        now += 1
    for c in enc_book:
        dec_book[enc_book[c]] = c
    return enc_book, dec_book

def dec(cipher):
    enc_book, dec_book = get_book(key)
    plaintext = ""
    for c in cipher.upper():
        if ord(c) not in all:
            plaintext += c
```

```
        else:
            plaintext += dec_book[c]
    print(plaintext)


if __name__ == '__main__':
    cipher = input("Input cipher:")
    dec(cipher)
```

WHUCTF{W0W_CRYPTOGRRRR4PHY_I5_S0O__IN7ER3ST1NG!HAHAHA}

WIERD AES

题目附件中给了加密代码，密钥，和 base64 之后的加密数据。

加密使用的是 AES 的 CBC（Cipher Block Chaining）模式，在这种模式中需要一个 IV（Initialization Vector）作为输入。一般在使用 AES 时，IV 是事前协商好的，只要加解密用的是同样的 IV，那么就可以正确解密数据。

但是本题中没有给出 IV，而是作为一个“secret”。

先研究一下 AES 中的各种加密模式，可以用搜索引擎搜索一下相关知识，这里给一个搜索到的链接：

> https://www.jianshu.com/p/79a225c2650e

根据 CBC 模式的加密流程反推一下 CBC 的解密过程。实际上，CBC 的解密过程中 IV 只对第 1 个 block 起作用；也就是说，在不使用 IV 的情况下，除去第 1 个 block，其它 block 也可是顺利解密；即使使用了错误的 IV，也只会导致第一个 block 出错。

于是可以随便选取一个 IV 值（比如说全零）先解密试一下，unpad 之后得到了许多条数据，但是这些数据的第一个 block 都不能保证是正确的。

观察一下容易发现这些数据的倒数第二个字符都是下划线，这很不寻常。每条数据最后一个字母拼起来之后倒序就是 flag。

提供一个 Python 脚本：

```python
from Crypto.Cipher import AES
import base64

from leaked_secret import key
AES_KEYSIZE = 32

def aes_unpad(s):
    n = s[-1]
    if 0 == n:
        n = AES_KEYSIZE
    return s[:-n]

def dec():
    f = open('ciphertext', 'r')
    ciphertext = f.readlines()
    f.close()
    ans = ''
    for i in range(len(ciphertext)):
        aes = AES.new(key, AES.MODE_CBC, b'0'*16)
        m = aes.decrypt(base64.b64decode(ciphertext[i]))
        m = aes_unpad(m)
        print(m)
        ans += chr(m[-1])
    print(ans[::-1])


if __name__ == '__main__':
    dec()
```

WHUCTF{I_H4T3_1nitia1izati0n_Vect0r!}

Not RSA

出题背景

此题主要考察对 RSA 加密、欧拉函数等相关信息安全数学基础知识的理解。

RSA

在 RSA 密码中：

$$N = p * q$$

$$\varphi(N) = (p-1) * (q-1)$$

$$ed \equiv 1 \, mod \, \varphi(N)$$

其中p和q均为质数，φ是欧拉函数，N和e为公钥，d为私钥。

明文m和密文c满足：

$$m^e \equiv c \, mod \, N$$

$$c^d \equiv m^{ed} \equiv m^{\varphi(N)} \equiv m \, mod \, N$$

读题

本题中给出了p、q、e、c。其中p、q是 128 比特的，$e = 65537$，c的长度为 256 比特。

第一反应是这像极了 RSA，于是一顿计算得到：

$$N = p * q$$

$$\varphi = (p-1) * (q-1)$$

$$d \equiv e^{-1} \, mod \, \varphi$$

$$m \equiv c^d \, mod \, N$$

却发现得到的m实在不像个 flag。

随便取一个不是0也不是1的明文m，如$m = 123$，用上面算出的d检验一下$m^{ed} \, mod \, N$是否等于1。果然不相等，这说明这个d不是与$N = p * q$相匹配的解密密钥，也即$\varphi = (p-1) * (q-1)$这个数值并不是N的欧拉函数值$\varphi(N)$。

解题

使用 yafu（https://sourceforge.net/projects/yafu/）等工具尝试分解一下题目给出的p和q，发现其中有一个不是质数，而是可以别分解为两个更小的质数。

对于三个质数a、b、c：

$$N = a * b * c$$

$$\varphi(N) = (a-1) * (b-1) * (c-1)$$

得到$\varphi(N)$后剩下的步骤就跟 RSA 没什么区别了。得到的明文是一个数字。有些平时不擅长做密码学题目的同学可能会卡在了这一步，却不知道该怎么转换为可读的字符串，已经很接近了！

有一种叫做“long to bytes”的转换方式，就是把数字转换为二进制，在高位补零使其比特长度达到 8 的倍数，然后将每 8 个比特作为 ASCII 码，得到对应的字符，拼起来即为一串字符串。这个功能在 Python 中可以通过 Crypto.Util.number.long_to_bytes、libnum.n2s 等函数实现。

WHUCTF{T0ld_ya_nOt-RSA.....}

PRISM

出题背景

本题包含的知识包括：基于 RSA 和 El Gamal 密码的 Kleptography、RSA 的“p-1”攻击。

出题灵感来自于以下论文

```
Young A, Yung M. The dark side of “black-box” cryptography or: Should we trust capstone?[C]//Annual International Cryptology Conference. Springer, Berlin, Heidelberg, 1996: 89-103.
```

El Gamal 密码

一种非常基本的，利用离散对数问题的困难性构造的公钥密码，这里用到的是模质数的群。

系统参数：大质数p，生成元g。

私钥：x。

公钥：$y = g^x \bmod p$。

加密：明文信息为M，选取随机数k，$c_1 = g^k \bmod p$，$c_2 = y^k * M \bmod p$，密文为$\langle c_1, c_2 \rangle$。

解密：$m = c_2 * (c_1^x)^{-1} \bmod p$。

Kleptography

Kleptography 是一种在密码实现过程中的留下后门的技术。

对于公钥密码系统，有公钥PK和私钥MK，PK是公开的，如果能够获取某PK对应的MK，则攻击成功。

Kleptography 的主要思路为，将足以暴露私钥MK的信息嵌入到PK中，同时不被密钥使用者或其他人察觉。“不被察觉”的方法也很简单，用**另一个密钥**对嵌入到PK中的信息进行加密即可。

如果这样的做成功了，那么，通讯的底层密码运算服务的算法提供者，就可以在不被发现的情况下，获取任何使用者的私钥，从而监听所有的通讯。

所谓“棱镜门”、“威胁国家安全”，不过如此。

分析题目

代码 keygen.py 有些长，但其实流程很清晰：①利用一个未知的 rsa_keygen()函数生成了 RSA 公钥(n_0, e_0)；②以(n_0, e_0)为输入，生成 RSA 公钥$(n_1 = p_1 * q_1, e_1)$；③以(n_1, e_1)为输入，生成 RSA 公钥$(n_2 = p_2 * q_2, e_2)$；④以(n_2, e_2)为输入，生成 El Gamal 公钥(p_3, g_3, y_3)；⑤以(p_3, g_3, y_3)为输入，生成 El Gamal 公钥(p_4, g_4, y_4)；⑥输出这 5 组公钥。

代码 enc.py 很简单，利用所有公钥中的最后一组，对数据进行 El Gamal 加密。

设这 5 组公钥各自对应的私钥分别为d_0, d_1, d_2, x_3, x_4。

逐个分析 keygen.py 的密钥生成过程。

步骤⑤：

$$b_4 = {y_3}^k * x_4 \bmod p_3$$

$$g_4 = {g_3}^k \bmod p_3$$

$$0 \le p_4 - b_4 < B_4$$

也即，用(p_3, g_3, y_3)加密了x_4，把加密结果存到了g_4, p_4中。若x_3是已知的，那么可以在B_4的时间

复杂度内得到x_4。

步骤④:

$$f_3 = FFF(x_3{}^{e_2} \bmod n_2, K_3 + i)$$

$$0 \le i < B_{3,1}$$

$$0 \le p_3 - f_3 < B_{3,2}$$

这里的函数 FFF 有两个输入数据，但其实可以把看作是对第一个参数进行的可逆哈希，且这种哈希是私有的，“私有”指的是第二个参数作为密钥。这一步用(n_2, e_2)加密了x_3，使用K_3作为密钥进行哈希之后存到了p_3中。若d_2是已知的，那么可以在$B_{3,1} * B_{3,2}$的时间复杂度内得到x_3。

步骤③:

$$p_{2,3} = GGG\left(\left(FFF(p_2, K_2 + i)\right)^{e_1} \bmod n_1, K_2 + j\right)$$

$$0 \le i < B_{2,1}$$

$$0 \le j < B_{2,2}$$

$$0 \le \left\lfloor \frac{n_2}{2^{1024}} \right\rfloor - p_{2,3} < B_{2,3}$$

这里的函数 GGG 与 FFF 同等看待。这一步对p_2使用K_2作为密钥进行 FFF 哈希，然后使用(n_1, e_1)加密，然后使用K_2作为密钥进行 GGG 哈希，结果存放到n_2的前一半比特位。若d_1是已知的，那么可以在$B_{2,1} * B_{2,2} * B_{2,3}$的时间复杂度内得到$p_2$，从而分解$n_2$得到$d_2$。

步骤②:

$$e_1 = p_1{}^{e_0} \bmod n_0$$

也即，用(n_0, e_0)加密了p_1，把加密结果存到了e_1中。若d_0是已知的，那么可以解密得到d_1。

步骤①。(n_0, e_0)的生成过程没有提供。这里的n_0是有弱点的，存在某非平凡（不等于 1 的）因子p，$p-1$不含大素因子。这种的n_0有些时候用 yafu 似乎也能成功分解。详细的攻击原理在下面给出。

解题过程: 先得到d_0，然后按顺序、每次利用前一个私钥，逐个破解d_1, d_2, x_3, x_4，最后用 El Gamal 的私钥x_4解密数据。

RSA 的“p-1”攻击

在 RSA 中，$N = p * q$，若

$$p - 1 = \prod_{i=1}^{t} p_i^{b_i}$$

且对于某个上界A，有:

$$p_i < A$$

那么，对于小于A的所有质数p_i，都可以找到一个a_i使其满足$p_i^{a_i} \le A$且$p_i^{a_i} > A$

计算

$$r = \prod_{i=1}^{k} p_i^{a_i}$$

那么显然有

$$p-1|r$$

又由费马小定理可知

$$g^{p-1} \equiv 1 \, mod \, p$$

那么可以推出

$$g^{r} \equiv 1 \, mod \, p$$

也即

$$p|g^{r}-1$$

因此，对于任选的整数g，通过计算

$$x \equiv g^{r}-1 \, mod \, n$$

$$p = gcd(x,n)$$

有很大概率可以得到n的非平凡因子p，N随即被分解。

解题脚本

```python
import random
import math
from Crypto.Util.number import long_to_bytes, bytes_to_long
from Crypto.Cipher import AES
from Crypto.Util import Counter
import gmpy2

def Generate_PrimeTable_Sieve(n):  # Eratosthenes
    l = list(range(1, n + 1))
    l[0] = 0
    for i in range(2, n + 1):
        if l[i - 1] != 0:
            for j in range(i * 2, n + 1, i):
                l[j - 1] = 0
    result = [x for x in l if x != 0]
    return result

def p_sub_1_attack(n, e):
    DIFFICULTY1, TRIAL_NUM = 20, 100
    A = 1 << DIFFICULTY1
    PrimesTable = Generate_PrimeTable_Sieve(A)
    r = []
    for prime in PrimesTable:
        a = math.floor(math.log(A, prime))
        r.append(prime ** a)
    p, q = 0, 0
    for _ in range(TRIAL_NUM):
        base = PrimesTable[random.randint(0, len(PrimesTable) - 1)]
        x = base
        for pa in r:
            x = pow(x, pa, n)
        if x != 1:
            p = gmpy2.gcd(x - 1, n)  # p = gcd(x-1,n)
            q = n // p  # q = n / p
            break
    phi = (p - 1) * (q - 1)
    d = gmpy2.invert(e, phi)
    return p, q, d

def FFF_1(food, key):
    K=0xe238c70fe2d1885a1b12debfa15484cab8af04675c39ff4c633d6177f234ed88
```

```
42.     key = long_to_bytes(key, 32)
43.     food = long_to_bytes(food, 128)
44.     aes = AES.new(key, AES.MODE_CTR, counter=Counter.new(128, initial_va
    lue=K))
45.     c = bytes_to_long(aes.decrypt(food))
46.     return c
47.
48.
49. def GGG_1(food, key):
50.     K=0xfd94d8de73e4aa8f4f452782b98a7870e82ec92a9db606fe4ca41f32d6df90c5
51.     K = long_to_bytes(K, 32)
52.     food = long_to_bytes(food, 128)
53.     aes = AES.new(K, AES.MODE_CTR, counter=Counter.new(128, initial_valu
    e=key))
54.     c = bytes_to_long(aes.decrypt(food))
55.     return c
56.
57.
58. def solve0(n, e):
59.     p, q, d = p_sub_1_attack(n, e)
60.     return n, d
61.
62.
63. def solve1(n, e, N, D):
64.     p = pow(e, D, N)
65.     assert 0 == n % p
66.     q = n // p
67.     phi = (p-1) * (q-1)
68.     d = gmpy2.invert(e, phi)
69.     return n, d
70.
71.
72. def solve2(n, e, N, D):
73.     Nbit = 1024
74.     nbit = 2048
75.     K =
    0xb6a022cd2fb960d4b6caa601a0412918fd80656b76c782fa6fe9cf50ef205ffb
76.     B1 = 8
77.     B2 = 8
78.     B3 = 1024
79.     ntop = n >> Nbit
80.     d = 0
81.     for k in range(0, B3):
82.         if k % 10 == 0:
83.             print("solve2: k =", k)
84.         p3 = ntop - k
85.         for j in range(0, B2):
86.             p2 = GGG_1(p3, K + j)
87.             p1 = pow(p2, D, N)
88.             for i in range(0, B1):
89.                 p = FFF_1(p1, K + i)
90.                 if 0 == n % p:
91.                     q = n // p
92.                     phi = (p - 1) * (q - 1)
93.                     d = gmpy2.invert(e, phi)
94.                     break
95.             if d > 0:
96.                 break
97.         if d > 0:
98.             break
```

```
    assert d > 0
    return n, d


def solve3(p, g, y, N, D):
    Nbit = nbit = 2048
    K = 0xfcec710a0313bb8f93e76e00ae6862b9be72dfd837db3b64ddde344bebfd2f50
    B1 = 8
    B2 = 1024
    x = 0
    for j in range(B2, -1, -1):
        if j % 100 == 0:
            print("solve3: j =", j)
        f = p - j
        for i in range(0, B1):
            x2 = FFF_1(f, K + i)
            maybe_x = pow(x2, D, N)
            if y == pow(g, maybe_x, p):
                x = maybe_x
                break
        if x > 0:
            break
    assert x > 0
    return p, g, y, x


def solve4(p, g, y, P, G, Y, X):
    Nbit = nbit = 2048
    B = 16384
    x = 0
    d = gmpy2.invert(pow(g, X, P), P)
    for i in range(0, B):
        if i % 100 == 0:
            print("solve4: i =", i)
        b = p - i
        maybe_x = b * d % P
        if y == pow(g, maybe_x, p):
            x = maybe_x
            break
    assert x > 0
    return p, g, y, x


def solve_all(keys, c1, c2):
    n, e = keys[0]
    N, D = solve0(n, e)
    print("0:", hex(D))

    n, e = keys[1]
    N, D = solve1(n, e, N, D)
    print("1:", hex(D))

    n, e = keys[2]
    N, D = solve2(n, e, N, D)
    print("2:", hex(D))

    p, g, y = keys[3]
    P, G, Y, X = solve3(p, g, y, N, D)
    print("3:", hex(X))
```

```
158.
159.     p, g, y = keys[4]
160.     P, G, Y, X = solve4(p, g, y, P, G, Y, X)
161.     print("4:", hex(X))
162.
163.     M = c2 * gmpy2.invert(pow(c1, X, P), P) % P
164.
165.     print(long_to_bytes(M))
166.
167. if __name__ == '__main__':
168.     keys = [ [
169.             0x8405381a3d5579319b4d9fd55224e9ce0b4091f67913c5b94c6ec15829b54e94a01a6d67dcf0fe1b411e9922ddde59bfe2cda9b36069455fd4e1b78ba21e91c532001daf951ab1af79fee2e6626beaa8855da8d6ababedc3632fdeaa9568a9c10f7396bf856753afa8641487db00ce854a5e01f00866306a93eff278f20d2645,
170.             0x10001, ],
171.         [
172.             0x8fd673cf7d987fa5451ac7e77687c841aef9d9b6109613cabce24d6f71c8ff89af5a69698a756c361649dade844e0238e069c857c0850b521810994eac2edce22be0f9fce279ec5e7aa3bcca41b3caa4d6ad8e07a1a5f825f9adccd5888384d92629309f8d962b57b81d16bc8d5c2c08fc5f951bf6c9f2d7c70974dead3e39b7,
173.             0x63d061be9751f43e223f10386e669312d62f1d11ecfc121b5b6ae99c38d114c86afd43071145953fefbb9b5a0da96e4e95db626449217d6a2529ba597dbf63cb814978782ac7b7e3453d173c6ce975b976917692820e52f8c4cb9fff0144790b88c1a7dffd717c70f8f6a2cfce362ba2002684b9812b6bb5fe4d7cdcaf968dc5, ],
174.         [
175.             0xcaec490cea0801761cef4fc444d998537f522e4e880e7ca73dc63ca7fd8cfc2310abccc5ff03618799216f591dd12db5405877a5dc1b066616007a95ff74082e0c5cc2738cf7e452110a88fc8e528b10a924a46ee0cd1237c6937ab0861784cd139c59bf5b01e436e7d4abb4f17a070d728dc16b71602e93c489ec0e26179eff769ca24d8a055fe4f1b00418cb68316d16bb66481eea5af2b0077869ce6e80dd5017c0591c77e7d44e5391d6d9b31252002936b66999375e7cf18fde53e7f652b8b036e68fe6cfb0cda307629d2e54ec400725c1fd99a51b23b52cd27698425d224ae8641f13e22564e9b5702e92a2fce8ca143d825cb4f04d93e03cb2ed64c9,
176.             0x10001, ],
177.         [
178.             0xcc52272693767db147968e81fd75b732d409d08dd347c0fdf0e878c7ce33b9557ed744156d2a889cc30f14603411a7993bd5f4edd6fd4b4d3303eec314f71575787c88a7718c3a27683f5a0e814770344099df4885a5f200aab8734e96d5dce1071e6b695f402a21f152eacdbdeb29ad2b64363472c20448c07bc1e0b9ccc837f096bcb4e7e36cef9901aa9abeeda92cf957498d21662c6cbf78ec1188d2ad7b5f1ba5e307ad5ee08843e41c69bbfc15bf683e8a5ea3f9f25543cd4d93349926c5845ef8c40de31e534b0142279359af2b8cb6ce03d6a2ff5d644ecb79ddb09ca2448419ed9e8d60b00c25a0f5d92d19367b3a8940f358f1f9dcf3f49648e6ed,
179.             0x547a6bacd423a7bb3b58232f125eb6004b72f89c249a154f858498a9a4c71a0c3851f7bd597761b86fdfa12e40390667a3cc9c051c83de1997829c68879c363b9dcc76e49dfd170926dd07b1cf5468c17232add79a531f27ce321bfb7a7c8946e2b1e9a3fa2815b40e2c537083b41085540838d361ac098aae14eddd20b59b554275afe17318aba9ba01ec9e53b6180b1dac5c3b6f587022e2f80ea304773d817875f70431623cc10e9b0ec004bac88c20014deacdc53fc39fe4524f3cf742e82e81f64f71d39d3cc5812bf0da93ecf929615e503c1ec6c7fe48a2659a04ae2ca3a798ed941a2d08ae4c6727d76cff334b1d36a701b72d93fac6524ea998ab47,
180.             0x7f4ebdebbe8a6161cec9a5ef9cce864666a6f3f8171aaace2dc0a9f3e765a5321d53f1ab7775f9a4b82bcfc5d7b52a0c9853ea018c30257b00a21f01ebaab556b06e2bad1dc5925759c529da3911552f9272fdac73a3a2c7b2871c877de866297eb8a74ec02873827afe8a122c00cd2945740682715bc6981fe49215740ea400d852ecb8273d4a826199a20ce3ef7a9212731e78a44d22a4c0a1cde2afaa844b332c21acc501433080cc81293aa00573ab88c6e8ffc302cefd739a74ea1c8fce5e9f88bb36a4b6845bf24479b2d7c8a4fe4f59932889fa3d43771e95ad330b5d2c593955e141aaf813441d6ce365b5eca12914db033605838aaed0db033c539c, ],
```

```
        [
            0xca15fd070931bc9ed037713c33859a51618ae7994e8b743a78d488431ac859774aa764df10b9d898f0ca5a8ea4f1ceb3b0e153d64e976410f7de524212088378ada1e5945e194bf305e7753816ad36d0ca4a768717bab626c73e2533233518e40e2c693c8a7344b588f978240af0eab18a9bf4d8efdc98b0cec607c4cae49e1a3750f74fcd3919929a0d16c78395c5b428a5bb3c86478712b8fd2c9389f9350ab07bcfd7da857d09767890e295c379f31498be07c8587ad8bb7bf20685655bd8eb82028d6698c1dc1c91ff4de624c8f01b2c0ef3bacb426865e19408d7807df2d38f33e4da83161468d36ef2be3e6f4a87105e899f670aa2aaf9c79dea58f527,
            0x395afa4e1881ea4437ea547c5ad16f379ec6bdf0188ca3e28626157ac6d4cc91200ee31fd3f0cf819c75341ea673ee1eb9c6e509cd9a004eda2f899b207e467d56aae201d18fc12142c4f33ce00d62169d413ad4cc6809fcaee959fb27260ea507859aa570ba1bed71f61b5be2716fa7600c25a1b6ab75b71ad64496bd0251be552d7968c82baf5d32f70fcae1285e4c971d0bcd4d7e069ef63f9e6f3deb90fdb6897e1d9aa08fbd3c78584514e5f132d48f3e432aea848fedb15904a2c0fe8b0ec3bae884726bc3d509cd97e43fb657a0fe0cae174429564bbeda1b37ea120addaa60b3db1c02903fed6b06e55deeae974074ceb080afd073dfbdfb5db9b25a,
            0x329da5818011ed7722285286281fdf0742008761b2d2a477df441e9cac23ad0e346e833729473a0a86c9b4425f772f6fba0d55dac438c979b4a7631ce07559829127ae53ed664403a9098a815fc0bc061280bed2594b7df06e80b8021300bbf7daf788191dcac6710523c18dca8c9567d1a32ea2c27b1d8d847b67f62d9250ced4e11b901b737c8b1da5bc2281d9e0057b39ae7f539e3daa146d2df02da6793efaec197361257315d75933259f826e0dffa036a38e4bd1314cabdc9e6eb7c21a7c23d3c1e7ee60914e7fac58a87e77e28c50875c6e4ab92156bd627ff14f1e27974268f575e546cb729743784a1c3f3fa6c6823a73e6726113a1d7558a19c588, ],
    ]
    c1 = 0x1f4e626597d94539c3b8df060ee00533babddc7155cb1c34b614df4b8aa4a0588fef27f5ee310380ec4f89be0dbe66d8e631aacd97647a0aa9ff3bdb3498b599e303fd5f891e03fa2de2f48967999db8c82f251fefe0ba2e1c212bc19325baa9ab3628896996
    9866ed6bd3c1fcec4669952bf6b00866ed1d88eb6e81da71f2a7069488c359b0c13b3464
    5efdc643e227d1ae0e8c012a26047395ccdff7e923e122fb97ff17fbe754f43d111c7468
    c5b34547cbbfb076b0831fd68c4582cbae4583365314da5a5e27caa2a4595de544e5d0df
    1ccaac998111c249604bbea76a6d1f8dae179353c3c56e2764927aa793b2a9fe3869a2fb
    746cd7a3a2acb1a753e0
    c2 = 0x4160a907ed70fc53c76bd6a074fa86b1f760ce5693136603b2d800ff7dc4
    3abd35401eea1884e96fd7c9dd999dab0ad53f9d4845e9d4435eeb54a14989330040fa31
    5796e2901fe06b967bfef535b2054a5598e1b9d032434417c9010763f73208e304e4cb91
    6c625f0e6ed70ff79893c6573097e685b998e62ad5fe434e4958542a9ebb059e97029722
    c372f5f8788f510880210f8f7d2a894ab2d6bc98c4407bc8ee30b3720720dd171372c7f7
    94c29b5d226ccec77ea5b15d10686fdfa21d401f3980247a0ac394f0133fbfe4fcef27af
    868540356e2d1c7a51b361ff29f72542c16e8367196b1f2082609510516520a5cf457753
    bef053673717ac1a6d7a
    solve_all(keys, c1, c2)
```

WHUCTF{B1G_8r0TH3R_i5_W4TCHIn9_U%^}

Root Hunt

出题背景

本题主要考查知识点为：在模N的群中，N不为质数且指数与$\varphi(N)$不互素的情况下求次方根。

本题的出题灵感来自于中国科学技术大学第六届信息安全大赛（hackergame2019）中的“密码学与数学”类别题目“十次方根”，这一比赛的相关信息可见于以下网址：

> https://github.com/ustclug/hackergame2019-writeups
> https://hack.lug.ustc.edu.cn/

（出题人当时并没有在比赛时间内做出这道题）

本想出一道不太难的关于信息安全数学基础知识运用的题目，结果一不小心设计得难度比预想大了不少。

本题有多种解法。（出题人觉得自己对相关知识的掌握也并不是非常扎实。）

读题

题目给出了一组n、e、c。这三个字母一摆，根据 RSA 中使用字母惯性，应该满足：

$$m^e \equiv c \bmod n$$

此题放出后不久，有同学跟我说，假定每次的m都不变的话，可以直接多次获取数据利用中国剩余定理计算m，但是这位同学这样做没有成功得到 flag。出题人刚听到这样的吐槽时其实也稍微有点慌，这个非预期解的难度可太低了。出题人立即跑去检查（自己写的，但已经忘了的）源码后发现，每次生成数据用到的明文m其实并不是 flag 本身，而是放了点随机数，所以直接跑中国剩余定理是解不出来的。

继续看题。数据中的e每次都是$0xd2$，也就是十进制的210，出题人选择这个数字是因为它是四个最小质数2、3、5、7的公倍数，这一条件造成的（严重）结果在后面会提到。

拿到一个模数，不管希望有多小，一定首先要用 yafu 等工具尝试分解一下。结果发现，这个n非常“不质数”，可以被分解为几十个小质数的乘积。但是这里面有很多相同的质数，互不相同的质数有 6 个，每个小质数的次数为 10 左右。可以得到n的质因数分解式：

$$n = P_1^{k_1} * P_2^{k_2} * P_3^{k_3} * P_4^{k_4} * P_5^{k_5} * P_6^{k_6}$$

欧拉函数：

$$\varphi(n) = \prod_{i=1}^{6}\left(P_i^{k_i-1} * (P_i - 1)\right)$$

遗憾的是，由于e的质因数构成，它往往与$\varphi(n)$是不互素的，也就不存在e关于$\varphi(n)$的数论倒数。

尝试降低指数

继续考虑，如果e可以分解为$e = a * b$，其中a与$\varphi(n)$可以不互素，但b与$\varphi(n)$是互素的，称a为e关于$\varphi(n)$的“不可逆部分”（un-invert-able），称b为e关于$\varphi(n)$的“可逆部分”（invert-able）；那么我们可以把$c = m^e$看作$(m^a)^b$，可以计算：

$$d \equiv b^{-1} \bmod \varphi(n)$$

$$m^a \equiv c^d \bmod n$$

这样我们可以得到$m^a \bmod n$的数值，明文m的指数从e降到了a，虽然a不是1（假如$a = 1$那么也就得到m了），但总还是把指数减小了一些，离目标近了一小步。但依然很遗憾的是，由于e的质因数构成实在是太特殊了，$\varphi(n)$往往是e的整数倍，因而e是没法在这里减小一些了。

尝试强行开根号

在 hackergame2019 的“十次方根”一题中，也有类似的模合数开根的情形。用户名为“ranwen”的同学在提交的公开 writeup 中，描述了一种有些巧妙的模开根的方法。“ranwen”同学的证明过程在如下链接指向的 pdf 文件中：

> https://github.com/ranwen/USTC-Hackergame2019-WP/blob/master/Root/Proof.pdf

出题人研究了这种方法，并决定将其稍微系统化一下。下面介绍这种方法。

已知条件：

$$m^e \equiv c \bmod n \tag{1}$$

其中，c、e、n、$\varphi(n)$已知，且$e|\varphi(n)$，求满足条件的m。

为了使m的指数变得与$\varphi(n)$互素，考虑加上某个数值X，同余式变为：

$$m^{e+X} \equiv c * m^X \bmod n \tag{2}$$

显然，这导致了同余号右侧多乘了一个m^X，把它消去的方法很简单，如果$X|\varphi(n)$，那么：

$$Y = \frac{\varphi(n)}{X}$$

$$m^{(e+X)*Y} \equiv c^Y * m^{XY} \equiv c^Y \bmod n \tag{3}$$

$m^{XY} = m^{\varphi(n)}$被消去是因为，任何数的$\varphi(n)$次方，模n之后，都为1。

再将两边分别开Y次方：

$$m^{e+X} \equiv c * g^{i*\frac{\varphi(n)}{Y}} \bmod n \tag{4}$$

其中，$i = 0,1,\dots,Y-1$；g是模n群的生成元，其阶为$\varphi(n)$。注意这里的**(4)**式是**(3)**式的充分不必要条件。

整理一下现有的条件：$e+X$与$\varphi(n)$互素；$X*Y=\varphi(n)$。这样就可以计算：

$$d \equiv (e+X)^{-1} \bmod \varphi(n)$$

$$m \equiv \left(c * g^{i*\frac{\varphi(n)}{Y}}\right)^d \bmod n$$

得到m了。

但要注意，**(3)**式到**(4)**式的转换会造成解集扩大，所以需要计算$m^e \bmod n$检验一下是否等于c。要想得到所有正确的m，需要将i从0遍历到$Y-1$，故考虑到计算复杂度，Y不可以过大。

模开根的方法介绍完毕。整体回看一下，其实它能够解决问题的限制条件还是相当多的。

继续遗憾的是，如果将这个方法直接应用到题目给出的n、e、c上，那么可用的Y值的大小往往是计算不能够承受的。

综合使用两种方法

回看已有的讨论，我们无法将指数e降低，这是因为$e|\varphi(n)$；我们无法直接应用上面的模开根的方法，是因为Y值过大，而Y值过大是因为$e+X=e+\frac{\varphi(n)}{Y}$与$\varphi(n)$互素的机会太小了。仔细想想，会发现其实这两个问题是由同一个障碍造成的，那就是$\varphi(n)$的质因数分解中包含了太多质数了，尤其

是小质数。

解决方案也很简单，虽然$\varphi(n)$很讨厌，但是$\varphi(P_i^{k_i}) = P_i^{k_i-1} * (P_i - 1)$友好多了。

P_i自身是较大的质数，不会与e产生"纠葛"，$(P_i - 1)$的质因数分解中一般也不会包含太多的小质数（随机生成质数p，观察$p-1$的质因数分解）。所以，先后利用上面讨论的两种方法（先分解$e = a * b$，将m的指数降为a，然后开a次方），用并不太大的复杂度，就可以得到不多于Y_i个m_i满足：

$$m_i{}^e \equiv c \bmod P_i^{k_i}$$

到这里，形势已经很明朗了。只需要再进行不多于

$$\prod_{i=1}^{6} Y_i$$

次的中国剩余定理的计算（CRT），就可以得到所有满足$m^e \equiv c \bmod n$的正确明文m。在它们中寻找带有 flag 标记的数据就可以了。

最后讲一个小地方，上面说的模开根方法的中间步骤中关于模n群的生成元g的选取，这里其实不必花太大精力验证g是否是生成元；为了大概率能得到所有正确的次方根，可以随机多选几个g，把每个g得到的开根结果m的集合取并集就好。

解题脚本

给出解题 Python 脚本：

```python
import gmpy2
from Crypto.Util.number import getRandomInteger, long_to_bytes
import itertools

def get_root(index, p, k, c):  # Find m such that m^index=c mod p^k
    # assert index < 1000
    print("p = " + hex(p))
    assert gmpy2.is_prime(p)
    assert p.bit_length() * k <= (2 << 16)

    n = p ** k
    phi = (p ** (k - 1)) * (p - 1)
    c %= n

    # First, split index into 2 parts, invert-able and un-invert-able.
    un_invert_able = gmpy2.gcd(index, phi)
    invert_able = index // un_invert_able
    while True:
        gcd = gmpy2.gcd(invert_able, phi)
        if gcd == 1:
            break
        invert_able //= gcd
        un_invert_able *= gcd
    assert invert_able * un_invert_able == index
    assert gmpy2.gcd(invert_able, phi) == 1
    print(invert_able, un_invert_able)

    # Get rid of the invert-able part.
    d = gmpy2.invert(invert_able, phi)
    c2 = pow(c, d, n)
    assert pow(c2, invert_able, n) == c
    print("c2 = " + hex(c2))

```

```
    # Next, find m such that m^un_invert_able=c2 mod p^k.
    Y = gmpy2.gcd(index, phi)
    X = phi // Y
    while True:
        gcd = gmpy2.gcd(X, un_invert_able)
        if gcd == 1:
            break
        X //= gcd
        Y *= gcd
    assert X * Y == phi
    assert gmpy2.gcd(un_invert_able + X, phi) == 1
    print("X = 0x%x,\nY = %d" % (X, Y))
    # Got a suitable Y.

    ans = set()
    counter = 0
    while True:
        g = gmpy2.next_prime(getRandomInteger(p.bit_length()) % p)
        for i in range(Y):
            m_uninv_and_X = c2 * pow(g, i * phi // Y, n) % n
            assert pow(m_uninv_and_X, Y, n) == pow(c2, Y, n)
            m = pow(m_uninv_and_X, gmpy2.invert(un_invert_able + X, phi)
, n)
            if pow(m, un_invert_able, n) == c2:
                ans.add(m)
        counter += 1
        if len(ans) >= un_invert_able or counter >= 10:
            break

    for m in ans:
        assert pow(m, index, n) == c
    return ans

def CRT(a_list, m_list, l):
    assert len(a_list) == len(m_list) == l > 0
    N = 1
    for m in m_list:
        N *= m
    ans = 0
    for i in range(l):
        Mi = N // m_list[i]
        ti = gmpy2.invert(Mi, m_list[i])
        ans = (ans + a_list[i] * ti * Mi) % N
    return ans


def work(index, n, p_list, k_list, c):
    l = len(p_list)
    assert len(k_list) == l
    m_list = []
    for i in range(l):
        m_list.append(p_list[i] ** k_list[i])
        assert gmpy2.is_prime(p_list[i])
    now_N = 1
    for i in range(l):
        now_N *= m_list[i]
    assert now_N == n

    root_list_list = []
    ans_num = 1
```

```
    for i in range(l):
        root_list = list(get_root(index, p_list[i], k_list[i], c))
        print("len(root_list) =", len(root_list))
        ans_num *= len(root_list)
        root_list_list.append(root_list)
    print("ans_num(total workload) =", hex(ans_num))
    counter = 0
    for a_list in itertools.product(*root_list_list):
        ans = CRT(a_list, m_list, l)
        counter += 1
        if counter % 0x1000 == 0:
            print("counter = " + hex(counter))
        if b'WHUCTF' in long_to_bytes(ans):
            print("counter = " + hex(counter))
            print(hex(ans), long_to_bytes(ans))
            return ans
    return 0

def solve():
    index = 210
    n = 0x1e7e40e80b73ee3ef68185c83c6dcd332a7b1783d44b6824a3f6ac1e5e55c4793d507f44b628fa7d7d7317ce174aba089463cc866e48ca0c357b1675030f31acc60b15740f3d1a329e7757b6a60521c05fcc724f05fa8320b91346b972be9ed971f725a4b28d12c8487f14aa584b1c7882f39c7a4a56ea214ff03209602f1bfed44e6bd7c41d4b552ad95532ff2ef0acf8df4f168249055bce5869c9ab1ee5c31de244712bd3ce79da4f278f9fa3c9263a2f92f2a1dbc2e5febb6a44487c8d479662928d0338f4b47a8b6c019f3847b1f24669ea394f89583f1620b9dd73715808491ce7e436eeab2f12e3bed8fc15b5f8e7c06cfb209d21f1e7a4f583b4cfc5193ec002f16d3caf4c6b4152f7a35efdfc5200947d2ec5940a7a6dceab7e67878847cc50d
    c = 0xe8327c3df503c84f79768c37a2a0cfa5883172fa6278bce54a1153318a8d4e8f14ac6fc9631f495a7467500fc9035bca3f1fc2b6e200f40f3562e048384e965e89adef49cc9a0c9bf85143a212594225abb9cb1bbdac1fcba7a02f785cc8a190cbdd3d4d3158b1571e98b8c7038d5cc0362dc5b67cfe8e268775714bd32b2b7c9ba4977d9ba4264983f9fbf3805b265d36145532219839a1bf3d0df67c7da889c1f2fb9ae465839b3592a50e93f2d58023723e94270b5b51ea27705c5c7ef9f4cb823cd81bb4a631d06e4c15d9074e3b8f6aef184e85104a5801ae9368fc0bd681db47324097a8562ecb64175c7ece17067543fc01b10edb9d833316aab154cbea361e1410d1be1a46c46c879f7fd800eb752cf77885ff791f40c535d9a3481e2004d6b4
    p_list = [
        81558703883,
        402917989931,
        23928329489,
        127206512233,
        7973312377,
        37359804007,
    ]
    k_list = [
        10, 9, 13, 8, 13, 13,
    ]
    ans = work(index, n, p_list, k_list, c)
    print(hex(ans))

if __name__ == '__main__':
    solve()
```

WHUCTF{^0^W0w_u_C4N_R3al1y_f1nd_r00T5_!}

Baby SM9

出题背景

本题的目的在于科普一下 IBE（Identity-Based Encryption，基于身份的加密、基于标识的加密）。我国的国密算法 SM9 就是一种 IBE。本题中呈现的 IBE 方案由论文：

```
Boneh D, Franklin M. Identity-based encryption from the Weil pairing[C]//Annual international cryptology conference. Springer, Berlin, Heidelberg, 2001: 213-229.
```

中的 Section 4 描述的“BasicIdent”修改而来，此论文时最早构造出成功 IBE 方案的论文。关于 IBE 的思想，有兴趣可以读下面 Shamir 的这篇十分古老的论文了解一下。

```
Shamir A. Identity-based cryptosystems and signature schemes[C]//Workshop on the theory and application of cryptographic techniques. Springer, Berlin, Heidelberg, 1984: 47-53.
```

（出题人觉得这篇上世纪八十年代的论文简洁明了地描述了这种非常优秀的想法，比起结构繁琐、绕来绕去的现代论文，可读性要强很多；但也不能否认，现代论文中大篇幅对于 Preliminary 知识的叙述，给了初入这个领域的人一些跟上前人步伐的机会。）

相比普通的公钥密码体系，IBE 的优点在于，不必获取目标用户的公钥，即可向其发送私密的信息。加密算法的输入中，公钥的位置被用户的 ID（或其衍生格式）取代。

说回“BasicIdent”，进行微小修改的原因主要在于，用不太长的 Python 脚本实现一个可读性较好的 Pairing（也叫 Bilinear Map，双线性映射）功能，（在出题人看来）比较困难。因此，出题人（偷了个懒）用另一种简单（但其实安全性不高）的方式实现了“双线性”的特性，并针对地对论文中的协议进行了一丢丢修改。

读题

题目首先要了一个电子邮箱地址，然后给出了一大把参数，包括公共参数、私钥、加密的信息。这里有个小坑，这些数据都是以八进制形式给出的。判别一个数的进制可以看其中有没有缺失的数字，比如这里的数据中没有看到数字 8 和 9，所以可以推测是八进制形式。

附件 scheme.py 中给出了公共参数的生成方式、私钥生成流程、加密函数。要根据数据的长度推测出一个关键变量 K 的大小，其实也就是模数N的比特长度，它会影响H_1和H_1这两个哈希函数的作用方式。即使不考虑 IBE，把这个题目单纯当作一个协议分析的题目，通过分析公共参数和加密流程得出解密方法，也是不太难的。

函数 setup()生成了主私钥（master key）和公共参数：

$$MK = \{s, \varphi(N)\}$$

$$params = \{N, P, P_{pub} = s * P \bmod \varphi(N), g\}$$

其中，N是一个 RSA 的模数（即，是两个大质数的乘积）；s, g, P的生成较为随机。

函数 extract()根据主私钥MK和用户的ID生成用户私钥d_{ID}：

$$Q_{ID} = H_1(ID)$$

$$d_{ID} = s * Q_{ID} \bmod \varphi(n)$$

函数 encrypt()根据用户的ID和公共参数$params$加密一条消息m：

$$Q_{ID} = H_1(ID)$$

$$g_{ID} = e(Q_{ID}, P_{pub})$$

$$U = g^{r*P} \bmod N$$

$$V = H_2({g_{ID}}^r \bmod N) \oplus m$$

得到(U,V)为密文，其中r为一次性的随机数，$\boldsymbol{e}$为“bilinear”运算，在这里被简单地定义为：

$$e(x,y) = g^{x*y} \bmod N$$

解题

整理一下，其实只需要算出$H_2({g_{ID}}^r \bmod N)$就可以了，即只需要算出${g_{ID}}^r \bmod N$即可。

$$\begin{aligned}{g_{ID}}^r &= \boldsymbol{e}\left(Q_{ID}, P_{pub}\right)^r \bmod N \\ &= (g^{Q_{ID}*s*P})^r \bmod N \\ &= (g^{r*P})^{Q_{ID}*s} \bmod N \\ &= U^{Q_{ID}*s} \bmod N \\ &= U^{d_{ID}} \bmod N\end{aligned}$$

因此

$$m = H_2(U^{d_{ID}} \bmod N) \oplus V$$

双线性运算

最后多说一嘴，真正的“双线性”运算是指：

设加法群G_1, G_2和乘法群G_T均为阶为q的循环群，且q为一大素数，P_1是G_1的生成元，P_2是G_2的生成元。定义如下形式的映射：

$$\boldsymbol{e}: G_1 \times G_2 \to G_T$$

称映射$\boldsymbol{e}$是双线性映射（或双线性对、Pairing），如果其满足如下三个条件：

①双线性：对于任意的$P \in G_1$，$Q \in G_2$和$a, b \in Z_n$，有$\boldsymbol{e}(aP, bQ) = \boldsymbol{e}(P,Q)^{ab}$；

②非退化性：$\boldsymbol{e}(P_1, P_2) \neq 1$，也即$\boldsymbol{e}(P_1, P_2)$是群$G_T$的生成元；

③可计算性：对于任意的$P \in G_1$，$Q \in G_2$，可以在多项式时间内有效计算$\boldsymbol{e}(P,Q)$。

实际用到的安全的双线性运算一般构建于椭圆曲线群上。

解题脚本

给出（相当简单的）Python 解题脚本：

```python
from Crypto.Util.number import long_to_bytes, bytes_to_long
import hashlib

K = 2048

def H2(x):
    ans = b''
    b = long_to_bytes(x, K // 8)
    now = 0
    while now * 8 < K:
        ans += hashlib.sha512(b[now:min(now+64, len(b)-1)]).digest()
        now = len(ans)
    return ans[:K]

def decrypt(c, d_ID, N):
    U, V = c
    m = long_to_bytes(V ^ bytes_to_long(H2(pow(U, d_ID, N))))
    return m

if __name__ == '__main__':
    N = 0o21254345037435216645726235615257215344466713570721155040602255
    35210063321754712424657207277753544723020115003516656451040117013362326
    74511047744700140037654704750253377052636306720673446756651467652651252
```

```
4234642103704104132455723321370141654634073712667472724737307143507523613145104577464124442157327650404143555103170040220412341643556012604766512370061105323704331616242717275116252735615453231135234414407736666563665744733342747351301264505704634024153466467026636325443261722250436716332202430645623442647454120072267433403033262744177612045136274071051144303515212361620774475074325316607440374646303574602672344560207751722765643667317047544044352230651622574420624125324 63
    d_ID = 0o6372124242207274553645604216057345425610746731223422163255153336347070066345600060111710430042430265705662715523620546772537361612360210201712555711563753555242513643666051231415720762741157037466124102661330460413555124503477716353225737664604675535735634767255721541343063347476221741112376377442131303271167372411701477654042026230272645222533354453501700700254202010556702342075311437242635772344176021013720764440132325615072647602103565746654026543122715025106714773171056151025715447355467775475645107142660270654032235130267473051340644535073536471022137756672341405301052304054654212025574043075475064345646615135033055510771461402324406303724417403213444364243375364437004
    U = 0o4240647757010421355377061523076674667145117444202511123544563664414701771317661744575111441234741524442632155653607470463010621673140433341562561100711000163411440717174547233054747265260504735525535702133353333102500012735423156656642463702447160737721102214064425310067570420466615013016007515072160463601071013503277075012011231514720126143315022424414471527726471207223052474647746525237106430003627464471755550405320673112766721137125570161733120544663147264055666500641026103545563675214626366537632672052344300320012772700425257336266144624242134672252623260011270236041077106514512034452107223522462416046702733547712466360513276776750305040701745622705112257702702530705536
    V = 0o3160356432322252105646317015470205421607056715002430002046307652440146144210653406016112447042363253643647644505622517313751566747261177516672461272200541555764646370137675535247712525674312716532606705475714516443741104302014153254114445067447753070525764571564302712634654441237026226166331275555701503062434624231362536114315547153474271021352600611622526213621640742011654661356105424132001556360413736613322362475064772041204644510523563107733713300555453473246125637201250305202064362130561126733701501167135300256261123137330767244531652456332012540472473575000207456511055037044747170765765503714466655256030644233326400264273764514227771440126226333404063071117734376512217 2
    c = U, V
    flag = decrypt(c, d_ID, N)
    print(flag)
```

WHUCTF{_Id_8aSeD_ENcr7PT1On_i5_W0nd3RfuLl1l1ll!!}

Bivibivi

前面的 Proof of Math 中是要求解一个简单的同余方程，其形式大概是：

$$a * x + b \equiv y \, mod \, N$$

解法并不难：

$$x \equiv (y - b) * a^{-1} \, mod \, N$$

其中a^{-1}是a关于N的数论倒数。

后面会要求给出某视频网站（www.bilibili.com）指定 av 号对应的 BV 号，和指定 BV 号对应的 av 号。前者非常容易实现，因为不久前该网站改为主要应用 BV 号，但还没有废除 av 号的访问功能，因此使用特定 av 号访问该网站，页面中会提供相应的 BV 号。后者实现起来就要略微麻烦一些。这可以看作一个对未知加密算法的选择明文攻击，加密算法的输入是 av 号，输出是 BV 号。

其实该网站给出了某些通过 av 号或 BV 号获取视频信息（包括 av 号和 BV 号）的 api，也就是说目前来看，即使不通过逆向二者互相转换的算法，也可以从该网站获取。出题人略有点担心大家把这个题做成一个 web 或者 misc，当然，这些方法也并不是被禁止的。

相信大多数解出此题的同学都是从某乎等平台获得了某些大神逆得到的 av 号和 BV 号互相转换的算法，和脚本，例如

```
https://www.zhihu.com/question/381784377/answer/1099438784
```

出题人（并不很努力地）自己尝试了一下逆向，但是没有成功得到有效的转换算法。

此题带有科普的成分。有兴趣的同学可以继续深挖一下类似的，根据输入和输出反推加密算法的方法。相关技能有可能被用于现代密码学中对 S 盒的攻击等。

WHUCTF{J0iN_us_4t_5pace_bilibi1i_com_474509521}

Proof of Work

此前没有见过或听说过 Proof of Work 的同学或许对此有些疑惑。

工作量证明（Proof of Work，PoW）是指，要求寻找出符合某个格式的明文m，使得$Hash(m)$等于特定值。其中$Hash$是一个安全的哈希函数，常见的有 SHA256、SHA512 等。MD5 已经被破解了，所以理论上它已经不安全了。

找到符号要求的m的方法只能依靠穷举所有可能的m，也即必然要进行大量的哈希运算。反过来讲，如果对方得到了这个m，那么可以认为对方一定进行过了特定数量的哈希计算。因此，对方提交了正确的m这件事，就可以用于证明，对方进行了一定的计算工作，因而被称为工作量证明。工作量证明机制被用于区块链中。

个人认为，在交互式题目的开头添加一个 PoW，或许有利于防止恶意访问、无意访问，从而节约服务器的运算资源。因为有些题目在生成数据的时候需要进行大量或相对大量的运算，比如 PRISM 一题。